Truck Platooning: week 3

Changyao Zhou, Jiaxin Pan

January 11, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by estimating the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week, we tried to fit the MPC model to the Carla simulator with the 'offline' mode. We recorded the data of the target vehicle by running with autopilot and computed the corresponding control values, including the throttle and the steering angle of the ego vehicle offline with the MPC model. To test the control value, we put them back into the Carla simulator, together with the recorded value of the target vehicle. After adjusting the MPC model and cost function multiple times, the slight errors of the MPC output still could not be eliminated.

To improve the output, we tried to use the 'online' mode this week. The global locations, yaw angles, and velocities of both target and ego vehicle are detected in each time step and the control values are computed based on these real-time inputs with MPC. With the 'online' mode, the real-time status can be detected every time step and the MPC model knows if its prediction works well.

All the videos are stored under https://syncandshare.lrz.de/getlink/fi8kj46iqdLggAzkQrG3PmfZ/week3.

3 Design the Cost Function

In the previous versions, we used the 'offline' mode of the MPC, which means we know the whole trajectory of the target vehicle. In the cost function, in each time step, not only the current state of the target vehicle is used to compute the cost, but also the known next 10 states of the target vehicle. With the known future states, the output is more reasonable. However, this is not achievable in real life for a vehicle chasing problem since the future states of the target vehicle should not be given.

With the 'online' mode, we are now solving the real vehicle chasing problem with only the current states of the target and ego vehicle given.

The horizon of our MPC model is set to be 10, which means in the cost function the 10 next states of the ego vehicle are predicted with the MPC model, as shown in equation (1), and then used to compute the cost with the corresponding future target states. In the 'offline' mode, the 'corresponding

future target states' mean the known future states of the target vehicle. In the 'online' mode, however, the future states are not given.

$$state = model(state, dt, u_{throttle}, u_{steering}) \tag{1}$$

One Way is to keep the future target states as the current target states, which means assuming the target vehicle static on the horizon of 10 time-steps. This may lead to the problem that the ego vehicle can never get as close to the target vehicle as desired since the real target vehicle is moving instead of keeping static.

To solve this problem, we tried to predict the future states of the target vehicle instead of assuming it is static. Since the throttle and steering angle of the target vehicle is unknown, we assumed that the yaw angle and the velocity of the target vehicle are stable in the next 10 steps.

$$x_{ref} = x_{ref} + v_{ref} * cos(yaw_{ref}) * dt \tag{2}$$
$$y_{ref} = y_{ref} + v_{ref} * sin(yaw_{ref}) * dt \tag{3}$$

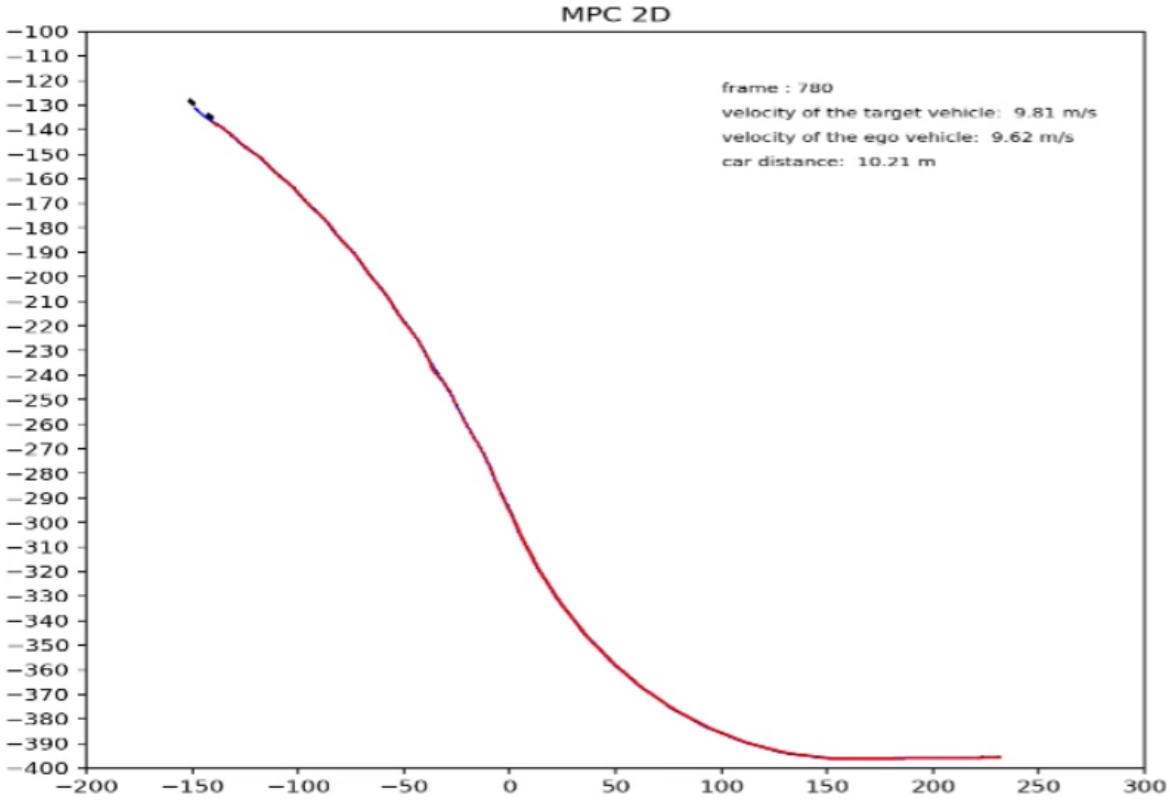

Figure 1: Screenshot of a test with Carla simulator.

The result of a test is shown in Figure 1, where blue dots represent the trajectory of the target vehicle and the red dots represent the trajectory of the ego vehicle. As can be seen from the figure, the two trajectories overlap almost at every step. This means that the MPC output control value is quite desirable.

4 Design the Plant Model

In the previous week, we tried to distinguish different gear statuses in the MPC plant model since the vehicle with different gear numbers behaves differently. However, with the 'online' mode, it seems unnecessary to distinguish different gears. One plant model can handle all different gears. Even though

the ego vehicle is not in the desired gear, the plant model can figure out the most reasonable control value.

5 Collect off-trajectory Data

To train our first fully-connected neural network, x- and y-positions, yaw angles, and velocities of both target and ego vehicles are used as inputs of the neural network.

The video of a normal trajectory can be checked here https://syncandshare.lrz.de/getlink/fi4tXmwuj38AFWuG76dvt5AT/on_trajectory.mp4.

Since the output control value may not be very perfect and lead the ego vehicle to unexpected positions, besides the 'on-trajectory' data, we also collect 'off-trajectory' data. This means, the MPC outputs are collected when the position of the ego vehicle is kind of off the ideal trajectory. With these training data, the network should learn how to deal with unexpected situations.

To collect the off-trajectory data, two different ways are used, namely the random impulses and random positions.

5.1 Off-trajectory data with random impulses

To apply random impulses on the ego vehicle, the Carla.actor.add_impulse [ADD]is used. Both the direction and the magnitude of the impulse are chosen randomly. The direction is set according to the global axis and can be either +x, -x +y, or -y. As shown in equations (4) - (6), the magnitude of the impulse is chosen from a range, which is computed based on the mass of the vehicle.

$$physics\_vehicle = vehicle.get\_physics\_control() \tag{4}$$

$$car\_mass = physics\_vehicle.mass \tag{5}$$

$$impulse = random.uniform(4.0, 7.0) * car\_mass \tag{6}$$

$$vehicle.add\_impulse(carla.Vector3D(impulse, 0, 0)) \tag{7}$$

The result is shown in Figure 2. In this test, a random impulse is generated and applied to the vehicle every 40 frames. The trajectory is much more noisy in comparison to the one without random impulses.

The video can be checked here https://syncandshare.lrz.de/getlink/fiLGpcowVC4A5V2ukmwUX1sz/off_impulse.mp4.

5.2 Off trajectory data with random positions

In addition to random impulse, we can also apply random translation and rotation in z-axis to put the ego-vehicle in a unexpected situation off the ideal trajectory. At every 40 frames, at time step t=20+40*n, the ego-vehicle will suddenly do a translation on the x, y axis or a rotation on the z axis (sudden change of yaw angle). The length of translation and the angle of rotation is randomly sampled in a specific range.

The resulting trajectory under random translation and rotation disturbances is shown in Figure 3. From the trajectory we could see the ego-vehicle could reach some points near the ideal trajectory. Also, the recovery of the path under disturbances shows the robustness of mpc controller.

The video can be checked here The video can be checked here https://syncandshare.lrz.de/getlink/fiPx1jxvWkjmzPZDj3zSxACp/off_translation_rotation.mp4.

6 Build the Neural Network

Until now, we have collected the dataset with 2400 samples, including 800 points exactly on the ideal trajectory and 1600 points off the ideal trajectory. Now we could start to build a NN with fully-connected layers. The NN has the x,y-location, yaw angle and velocity of both target vehicle and ego-vehicle as input. And the output are two control values, throttle and steering angle. We have built a simple model which includes four linear layers with 128, 256, 256, 64 neurons respectively. And we just started to train the NN and haven't got a stable result.

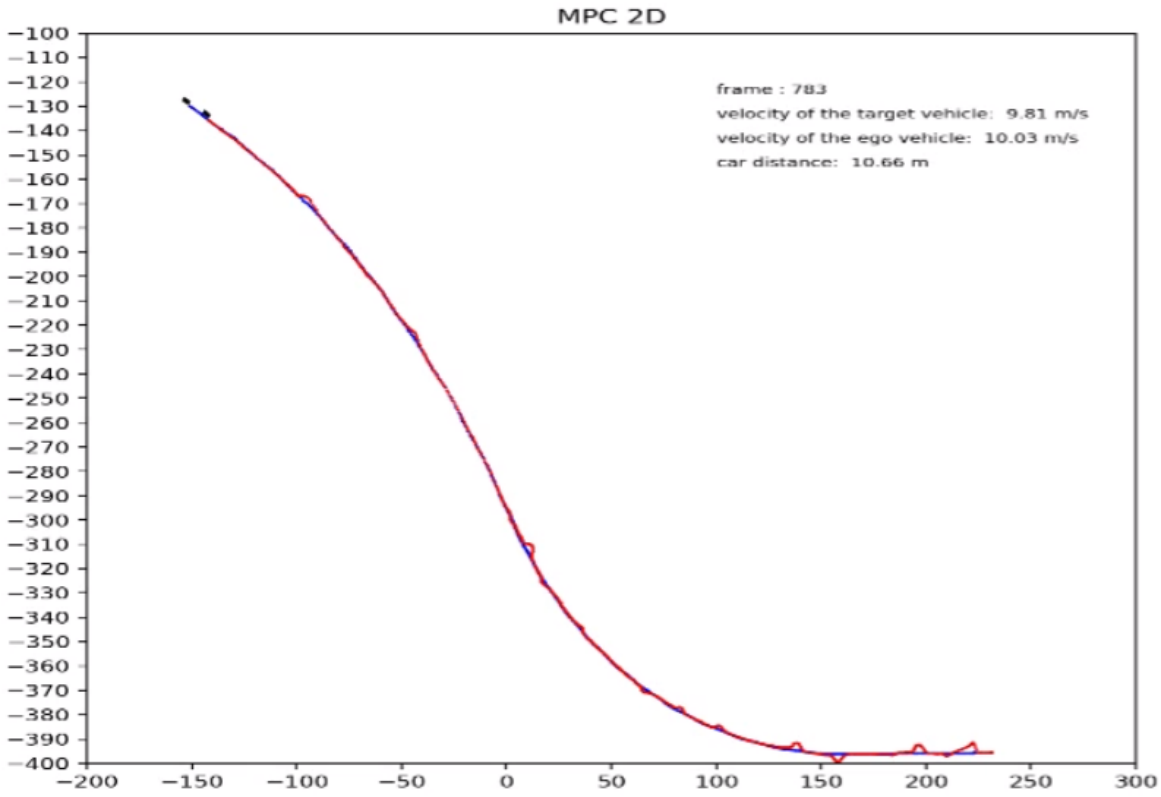

Figure 2: Off trajectory with random impulses.

7 Conclusion and future work

In this week, We tried to apply the MPC model to the Carla simulator with 'online' mode. Instead of knowing the whole trajectory in advance and predicting the control values offline, only the current values are used as input of the MPC. As an outcome, the MPC outputs are more desirable with the 'online' mode since the MPC can keep tracking the latest status.

To train our first fully-connected Neural Network, we collected both on-trajectory and off-trajectory data to let the network learn how to deal with unexpected situations.

In the next week, we will start training the first network. If it works well, we will change the inputs of the network step by step.

References

[ADD] Carla Add Impulse. https://github.com/carla-simulator/carla/blob/8854804f4d7748e14d937ec763a2912823a7e5f5/Docs/python_api.md#method.

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.

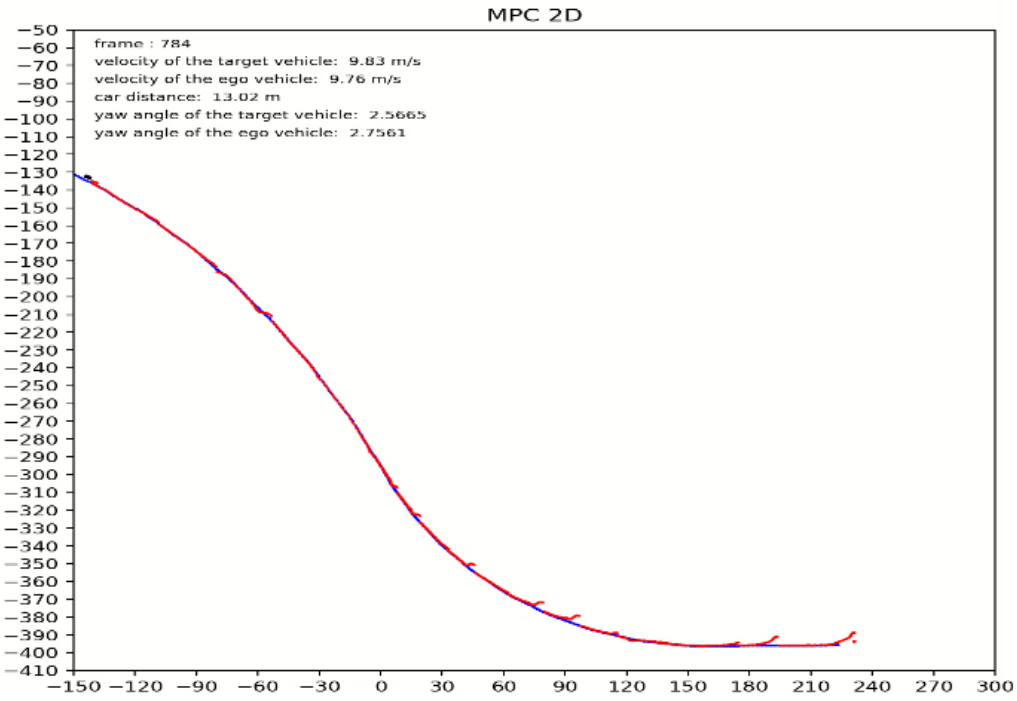

Figure 3: Off trajectory with random translation and rotation.